**Designing an OO Program**

In this reading, I will show you how to create classes in JavaScript, using all the concepts you've learned so far.

Specifically, I'm preparing to build the following inheritance hierarchy:

Animal / \ Cat Bird / \ \ HouseCat Tiger Parrot

There are two keywords that are essential for OOP with classes in JavaScript.

These keywords are *extends* and *super*.

The *extends* keyword allows me to inherit from an existing class.

Based on the above hierarchy, I can code the *Animal* class like this:

class Animal {

    // ... class code here ...

}

Then I can code, for example, the Cat sub-class, like this:

class Cat extends Animal {

    // ... class code here ...

}

This is how the extends keyword is used to setup inheritance relationships.

The super keyword allows me to "borrow" functionality from a super-class, in a sub-class. The exact dynamics of how this works will be covered later on in this lesson.

class Animal {

    // constructor: color, energy

    // isActive()

        // if energy > 0, energy -=20, console log energy

        // else if energy <= 0, sleep()

    // sleep()

        // energy += 20

        // console.log energy

}

class Cat extends Animal {

    // constructor: sound, canJumpHigh, canClimbTrees, color, energy

    // makeSound()

        // console.log sound

}

class Bird extends Animal {

    // constructor: sound, canFly, color, energy

    // makeSound()

        // console.log sound

}

class HouseCat extends Cat {

    // constructor: houseCatSound, sound, canJumpHigh, canClimbTrees, color, energy

    // makeSound(option)

        // if (option)

            // super.makeSound()

        // console.log(houseCatSound)

}

class Tiger extends Cat {

    // constructor: tigerSound, sound, canJumpHigh, canClimbTrees, color, energy

    // makeSound(option)

        // if (option)

            // super.makeSound()

        // console.log(tigerSound)

}

class Parrot extends Bird {

    // constructor: canTalk, sound, canJumpHigh, canClimbTrees, color, energy

    // makeSound(option)

        // if (option)

            // super.makeSound()

        // if (canTalk)

            // console.log("talking!")

}

This is how the extends keyword is used to setup inheritance relationships.

The super keyword allows me to "borrow" functionality from a super-class, in a sub-class. The exact dynamics of how this works will be covered later on in this lesson.

Now I can start thinking about how to implement my OOP class hierarchy.

Before I even begin, I need to think about things like: \* What should go into the base class of Animal? In other words, what will all the sub-classes inherit from the base class? \* What are the specific properties and methods that separate each class from others? \* Generally, how will my classes relate to one another?

Once I've thought it through, I can build my classes.

So, my plan is as follows:

1. The Animal class' constructor will have two properties: color and energy

2. The Animal class' prototype will have three methods: isActive(), sleep(), and getColor().

3. The isActive() method, whenever ran, will lower the value of energy until it hits 0. The isActive() method will also report the updated value of energy. If energy is at zero, the animal object will immediately go to sleep, by invoking the sleep() method based on the said condition.

4. The getColor() method will just console log the value in the color property.

5. The Cat class will inherit from Animal, with the additional sound, canJumpHigh, and canClimbTrees properties specific to the Cat class. It will also have its own makeSound() method.

6. The Bird class will also inherit from Animal, but is own specific properties will be quite different from Cat. Namely, the Bird class will have the sound and the canFly properties, and the makeSound method too.

7. The HouseCat class will extend the Cat class, and it will have its own houseCatSound as its special property. Additionally, it will override the makeSound() method from the Cat class, but it will do so in an interesting way. If the makeSound() method, on invocation, receives a single option argument - set to true, then it will run super.makeSound() - in other words, run the code from the parent class (Cat) with the addition of running the console.log(this.houseCatSound). Effectively, this means that the makeSound() method on the HouseCat class' instance object will have two separate behaviors, based on whether we pass it true or false.

8. The Tiger class will also inherit from Cat, and it will come with its own tigerSound property, while the rest of the behavior will be pretty much the same as in the HouseCat class.

9. Finally, the Parrot class will extend the Bird class, with its own canTalk property, and its own makeSound() method, working with two conditionals: one that checks if the value of true was passed to makeSound during invocation, and another that checks the value stored inside this.canTalk property.

Now that I have fully explained how all the code in my class hierarchy should work I might start implementing it by adding all the requirements as comments inside the code structure.

At this stage, with all the requirements written down as comments, my code should be as follows:

Now that I've coded my requirements inside comments of otherwise empty classes, I can start coding each class as per my specifications.

### Coding the Animal class

First, I'll code the base Animal class.

class Animal {

    constructor(color = 'yellow', energy = 100) {

        this.color = color;

        this.energy = energy;

    }

    isActive() {

        if(this.energy > 0) {

            this.energy -= 20;

            console.log('Energy is decreasing, currently at:', this.energy)

        } else if(this.energy == 0){

            this.sleep();

        }

    }

    sleep() {

        this.energy += 20;

        console.log('Energy is increasing, currently at:', this.energy)

    }

    getColor() {

        console.log(this.color)

    }

}

Each animal object, no matter which one it is, will share the properties of color and energy.

Now I can code the Cat and Bird classes:

class Cat extends Animal {

    constructor(sound = 'purr', canJumpHigh = true, canClimbTrees = true, color, energy) {

        super(color, energy);

        this.sound = sound;

        this.canClimbTrees = canClimbTrees;

        this.canJumpHigh = canJumpHigh;

    }

    makeSound() {

        console.log(this.sound);

    }

}

class Bird extends Animal {

    constructor(sound = 'chirp', canFly = true, color, energy) {

        super(color, energy);

        this.sound = sound;

        this.canFly = canFly;

    }

    makeSound() {

        console.log(this.sound);

    }

}

Note: If I didn't use the super keyword in our sub-classes, once I'd run the above code, I'd get the following error: Uncaught ReferenceError: Must call super constructor in derived class before accessing 'this' or returning from derived constructor.

And now I can code the three remaining classes: HouseCat, Tiger, and Parrot.

Note: If I didn't use the super keyword in our sub-classes, once I'd run the above code, I'd get the following error: Uncaught ReferenceError: Must call super constructor in derived class before accessing 'this' or returning from derived constructor

And now I can code the three remaining classes: HouseCat, Tiger, and Parrot.

class HouseCat extends Cat {

    constructor(houseCatSound = "meow", sound,canJumpHigh,canClimbTrees, color,energy) {

        super(sound,canJumpHigh,canClimbTrees, color,energy);

        this.houseCatSound = houseCatSound;

    }

    makeSound(option) {

        if (option) {

            super.makeSound();

        }

        console.log(this.houseCatSound);

    }

}

class Tiger extends Cat {

    constructor(tigerSound = "Roar!", sound,canJumpHigh,canClimbTrees, color,energy) {

        super(sound,canJumpHigh,canClimbTrees, color,energy);

        this.tigerSound = tigerSound;

    }

    makeSound(option) {

        if (option) {

            super.makeSound();

        }

        console.log(this.tigerSound);

    }

}

class Parrot extends Bird {

    constructor(canTalk = false, sound,canFly, color,energy) {

        super(sound,canFly, color,energy);

        this.canTalk = canTalk;

    }

    makeSound(option) {

        if (option) {

            super.makeSound();

        }

        if (this.canTalk) {

            console.log("I'm a talking parrot!");

        }

    }

}

Now that we've set up this entire inheritance structure, we can build various animal objects.

For example, I can build two parrots: one that can talk, and the other that can't.

var polly = new Parrot(true); // we're passing `true` to the constructor so that polly can talk

var fiji = new Parrot(false); // we're passing `false` to the constructor so that fiji can't talk

polly.makeSound(); // 'chirp', 'I'm a talking parrot!'

fiji.makeSound(); // 'chirp'

polly.color; // yellow

polly.energy; // 100

polly.isActive(); // Energy is decreasing, currently at: 80

var penguin = new Bird("shriek", false, "black and white", 200); // setting all the custom properties

penguin; // Bird {color: 'black and white', energy: 200, sound: 'shriek', canFly: false }

penguin.sound; // 'shriek'

penguin.canFly; // false

penguin.color; // 'black and white'

penguin.energy; // 200

penguin.isActive(); // Energy is decreasing, currently at: 180

Also, I can build a pet cat:

var leo = new HouseCat();

Now I can have leo purr:

// leo, no purring please:

leo.makeSound(false); // meow

// leo, both purr and meow now:

leo.makeSound(true); // purr, meow

Additionally, I can build a tiger:

cuddles.makeSound(false); // Roar!

cuddels.makeSound(true); // purr, Roar!

class Animal {

    constructor(color = 'yellow', energy = 100) {

        this.color = color;

        this.energy = energy;

    }

    isActive() {

        if(this.energy > 0) {

            this.energy -= 20;

            console.log('Energy is decreasing, currently at:', this.energy)

        } else if(this.energy == 0){

            this.sleep();

        }

    }

    sleep() {

        this.energy += 20;

        console.log('Energy is increasing, currently at:', this.energy)

    }

    getColor() {

        console.log(this.color)

    }

}

class Cat extends Animal {

    constructor(sound = 'purr', canJumpHigh = true, canClimbTrees = true, color, energy) {

        super(color, energy);

        this.sound = sound;

        this.canClimbTrees = canClimbTrees;

        this.canJumpHigh = canJumpHigh;

    }

    makeSound() {

        console.log(this.sound);

    }

}

class Bird extends Animal {

    constructor(sound = 'chirp', canFly = true, color, energy) {

        super(color, energy);

        this.sound = sound;

        this.canFly = canFly;

    }

    makeSound() {

        console.log(this.sound);

    }

}

class HouseCat extends Cat {

    constructor(houseCatSound = "meow", sound,canJumpHigh,canClimbTrees, color,energy) {

        super(sound,canJumpHigh,canClimbTrees, color,energy);

        this.houseCatSound = houseCatSound;

    }

    makeSound(option) {

        if (option) {

            super.makeSound();

        }

        console.log(this.houseCatSound);

    }

}

class Tiger extends Cat {

    constructor(tigerSound = "Roar!", sound,canJumpHigh,canClimbTrees, color,energy) {

        super(sound,canJumpHigh,canClimbTrees, color,energy);

        this.tigerSound = tigerSound;

    }

    makeSound(option) {

        if (option) {

            super.makeSound();

        }

        console.log(this.tigerSound);

    }

}

class Parrot extends Bird {

    constructor(canTalk = false, sound,canFly, color,energy) {

        super(sound,canFly, color,energy);

        this.canTalk = canTalk;

    }

    makeSound(option) {

        if (option) {

            super.makeSound();

        }

        if (this.canTalk) {

            console.log("I'm a talking parrot!");

        }

    }

}

var fiji = new Parrot(false); // we're passing `false` to the constructor so that fiji can't talk

var polly = new Parrot(true); // we're passing `true` to the constructor so that polly can talk

fiji.makeSound(); // undefined

fiji.makeSound(true); // chirp

polly.makeSound(); // I'm a talking parrot!

polly.makeSound(true); // chirp, I'm a talking parrot!

polly.color; // yellow

polly.energy; // 100

polly.isActive(); // Energy is decreasing, currently at: 80

var penguin = new Bird("shriek", false, "black and white", 200); // setting all the custom properties

penguin; // Bird {color: 'black and white', energy: 200, sound: 'shriek', canFly: false }

penguin.sound; // 'shriek'

penguin.canFly; // false

penguin.color; // 'black and white'

penguin.energy; // 200

penguin.isActive(); // Energy is decreasing, currently at: 180

var leo = new HouseCat();

// leo, no purring please:

leo.makeSound(false); // meow

// leo, both purr and meow now:

leo.makeSound(true); // purr, meow

var cuddles = new Tiger();

cuddles.makeSound(false); // Roar!

cuddles.makeSound(true); // purr, Roar!